List Comprehensions

in Python

Jordi Gomez Bao

Lists in Python

- From some elements
  - mylist = [0,1,2,3,4]
- From a function
  - mylist = range(0,5)
- From an iterable
  - mylist = list(myiterable)
- From a **list comprehension**
  - mylist = [i | for i in myiterable | if condition]
- From a generator expression
  - mylist = list(i | for i in myiterable | if condition)

List Comprehension

Syntaxis to make new lists where each element is the result of some operations applied to each member of another iterable.

http://docs.python.org/2/tutorial/datastructures.html

```
mylist = []
for i in range(0,5):
    mylist.append(i)
mylist
[0, 1, 2, 3, 4]
```

```
squarelist = []
for i in range(0,5):
    if i%2==0:
        squarelist.append(i*i)
squarelist
[0, 4, 16]
```

```
mylist = [i for i in range(0,5)]
mylist
[0, 1, 2, 3, 4]
```

```
squarelist = [i*i for i in range(0,5)if i%2==0]
squarelist
[0, 4, 16]
```

Advantages

- Better
  - Readability
  - Compactness
  - Expressivity
- Faster execution

```
timeit.timeit('loop_matrix(100)',
  'from matrix import loop_matrix',
  number=1000)
1.660736083984375
timeit.timeit('list_comprehension_matrix(100)',
  'from matrix import list_comprehension_matrix',
  number=1000)
1.0217621326446533
```

matrix.py

```
import random

def loop_matrix(size):
    matrix = []
    rows = columns = range(0, size)
    for i in rows:
        row = []
        for j in columns:
            row.append(i*j)
        matrix.append(row)
    return matrix

def list_comprehension_matrix(size):
    rows = columns = range(0, size)
    return [[i*j for j in columns] for i in rows]
```

Spelling Corrector

http://norvig.com/spell-correct.html

```
import re, collections
NWORDS = train(words(file('big.txt').read()))
alphabet = 'abcdefghijklmnopqrstuvwxyz'

def words(text):
    return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model
```

```
def edits1(word):
   splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in
NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word))
                or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

Conclusion

- List with elements computed from expression
- Better (readability, expressivity and compactness) and faster (execution)
- Pythonic way
- Not possible to use them when elements
  - involve other data structures
  - are computed by complex rules